



CONFERENCE

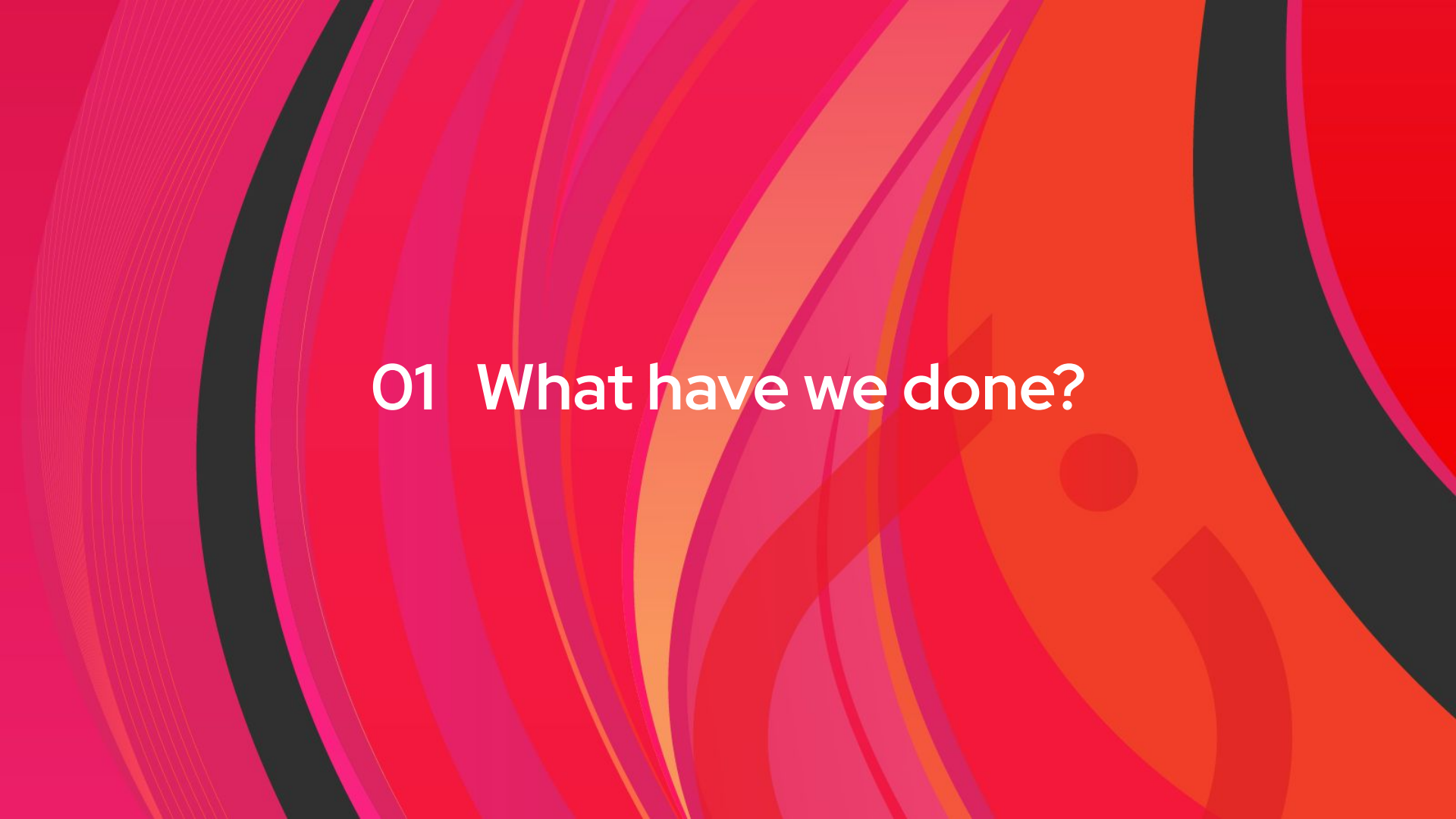
—NORTH AMERICA 2026—

Native DSL Operators in PyTorch Core

Simon Layton
Pytorch

Agenda

1. What have we done?
2. Why did we do it?
3. How was it done?
4. Results (or: why should you care?)



01 What have we done?

What did we do?

- **Allow devs to implement operators in-tree in PyTorch using python**
 - Core ATen ops
 - DSLs
 - Both Just-In-Time (JIT) & Ahead-of-time (AOT)

Supported DSLs (for now)

- **cuteDSL**
 - JIT & AOT
- **Triton**
 - JIT, first AOT op under review
- **flyDSL**
 - JIT
- **Helion**
 - JIT



Built to be DSL-agnostic & extensible
If you have a DSL you want adding, talk to me!

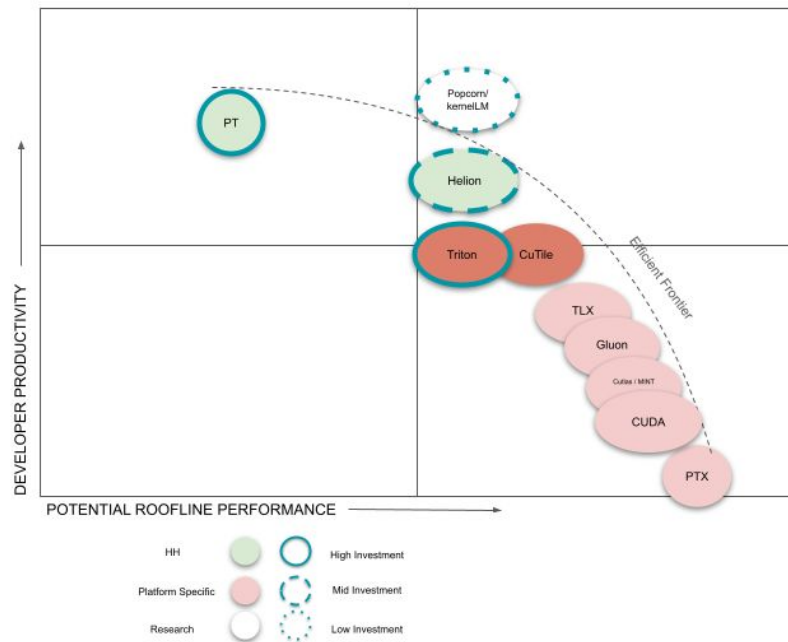
01 What



02 Why did we do this?

Why is this good for us?

- **Leverage large + growing python-based ecosystem**
 - ~17 (inc. this one!) talks over these 2 days
- **Enable high-level kernel authoring**
 - Triton, cuteDSL, Helion, cuTile, FlyDSL, etc.
 - Don't need to write CUDA-C++
 - Still get SoTA performance
 - Fast iterations for debug/tuning



Why is this good for you?

- **SoTA Implementations now more commonly in DSLs**
 - FAv4 (Tri Dao et. al)
 - MSLK, TBD
- **You can get your hands on these with less latency**
 - No need for anyone to translate to C++...
- **Less hassle managing packages**
 - Torch will just work™

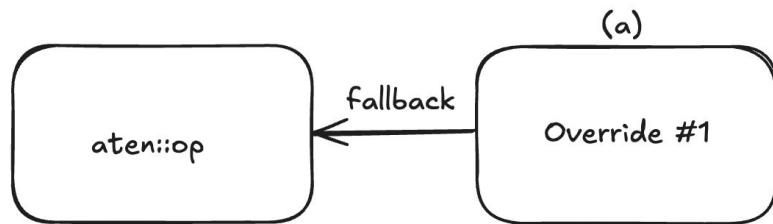
03 How did we do it?

High-Level

- **Two Regimes**
 - **Just-In-Time (JIT)**
 - Pure python logic & implementation
 - **Ahead-Of-Time (AOT)**
 - Embed compiled DSL code into libtorch*.so
 - Python & C++
 - Some auto-generated, some manual (or agentic)

How: JIT

- **Via. Operator overrides.**
 - Interacts w/dispatcher
 - Fall back to ATen
- **Provide infrastructure to help**
- **Pure python - no C++**



Provide 2 things for an override:
cond – *When the override should fire*
impl – *What the override should do*

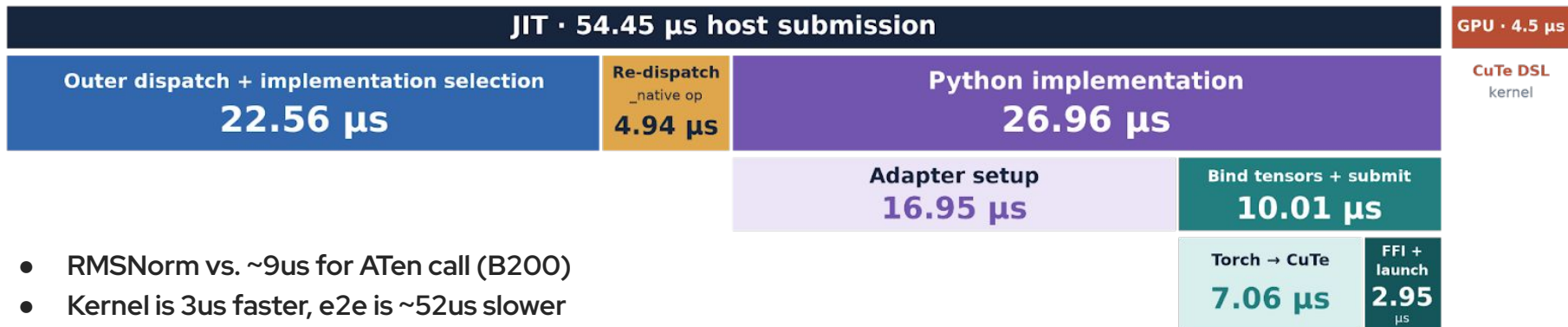
Why JIT?

- **Low cost-of-entry**
 - Very simple - 2x python functions + register
- **Runtime specialization**
 - Use your call's inputs to generate better kernel
- **Large cross-product of options**
 - Many kernels / specializations
 - Large cross-product of options, only need a few at a time
 - Re-use the same case (minimal recompiles)

JIT: Disadvantages

RMSNorm: JIT host submission

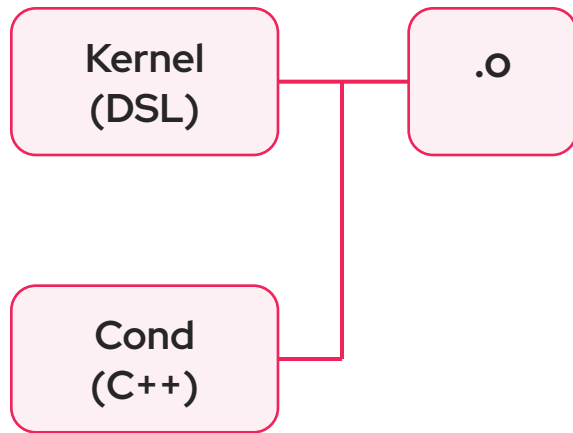
1M×7168 · warm eager · cache hit



- RMSNorm vs. ~9us for ATen call (B200)
- Kernel is 3us faster, e2e is ~52us slower
 - Low kernel runtime - no amortization
 - Overhead is ~fixed for input size
- Note: CUDAGraphs do remove all of this :)

Why AOT?

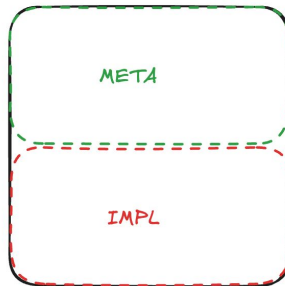
- **Lower overheads**
 - Know / can enumerate specializations ahead of time
 - Bake a lot of the slow logic into C++
 - Avoid most of the python... stuff.
- **Perfect for short-running kernels**
 - Overheads would dominate
 - Pointwise, reductions, top-k, etc.



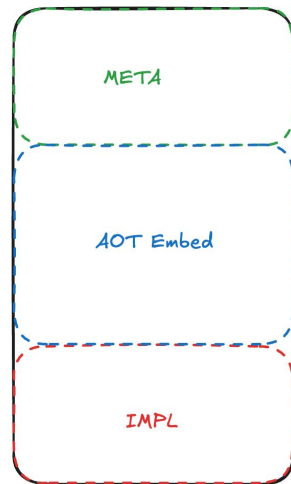
How: AOT

- **Insert guards & calls into generated code**
 - Call AOT if possible
 - Fallback to ATen
- **Still use some python dispatch**
 - Needed for hybrid
 - Fast “AOT-covered?” methods

C++ generated structured op



C++ generated structured op
w/AOT embed

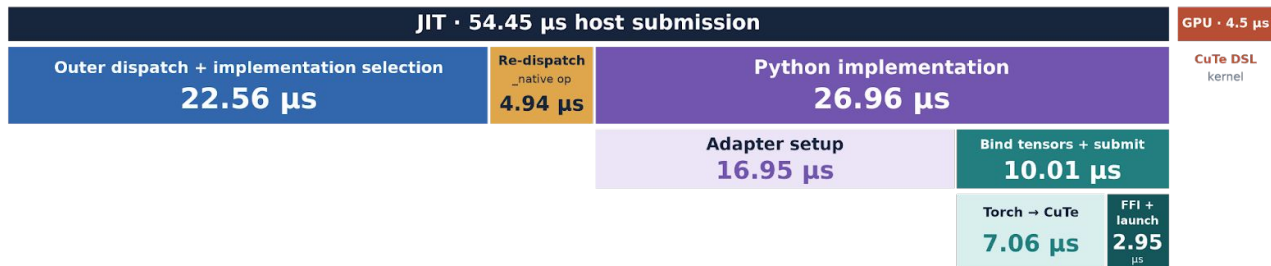


Opt-in: No need for AOT if you don't want to!

AOT: Advantages

RMSNorm: JIT, AOT and ATen

1Mx7168 · warm eager · cache hit



AOT: additional host overhead + GPU

Same time scale



ATen: 8.2 μs end-to-end

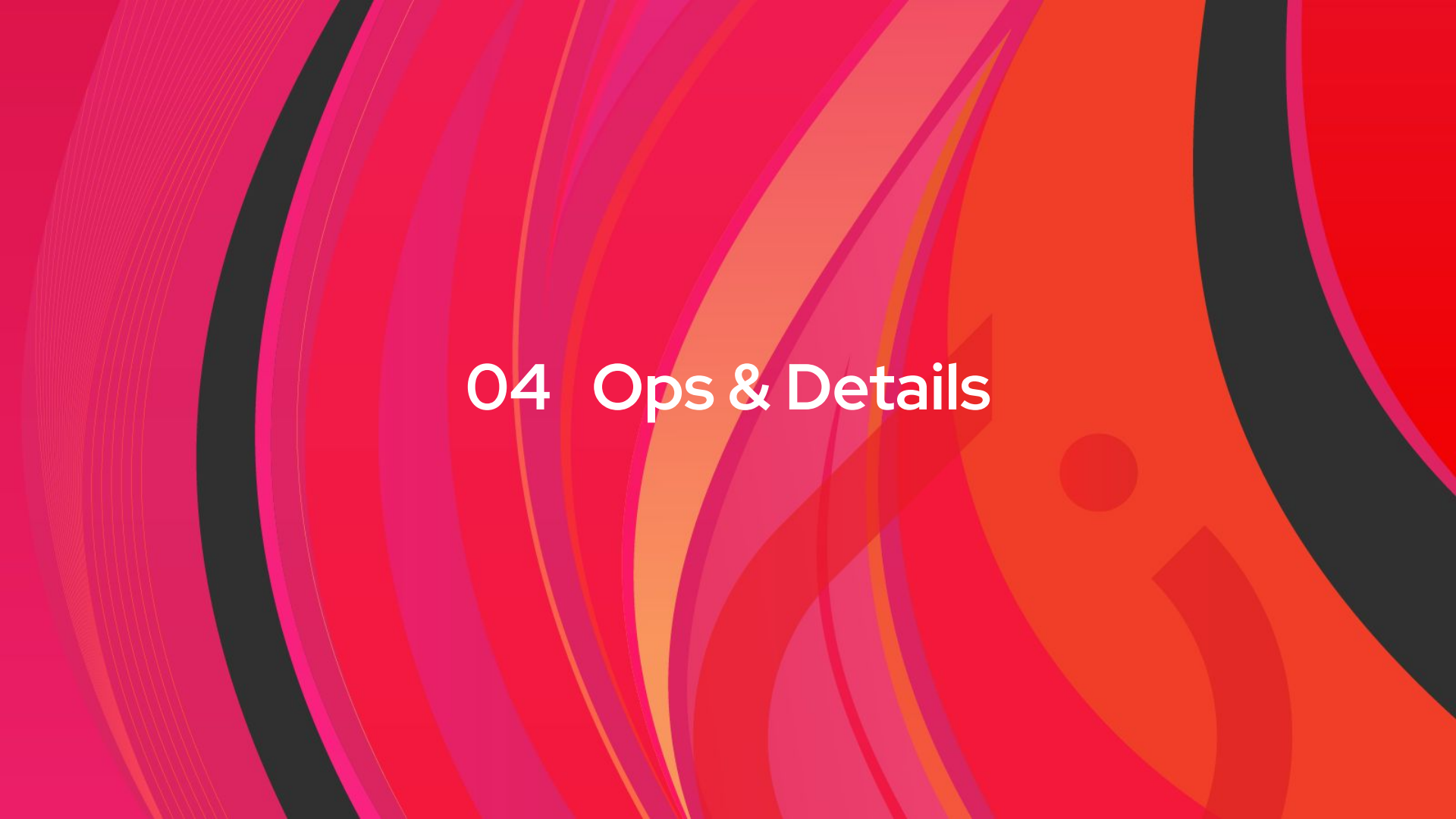
Same time scale



- RMSNorm vs. ~9us for ATen call (B200)
- Kernel is 3us faster, e2e is ~52us slower
 - Low kernel runtime – no amortization
 - Overhead is ~fixed

Hybrid – Let's have the best of both worlds!

- **JIT & AOT better at different times**
 - Long-running: host overhead less interesting, leave as JIT
 - Short-running: host overhead dominates, move to AOT
- **Save binary size**
 - AOT common dtypes/inputs/sizes
 - JIT the rest

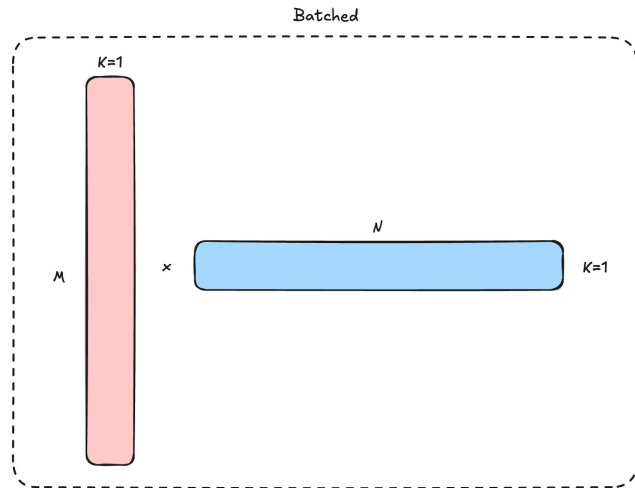


04 Ops & Details

Use Case #1 – Perf spot-fix

- **BMM Outer-product**

- Only supports $K = 1$
- cuBLAS perf is horrible
 - Triton is 6-17x faster on bf16 cases we care about
 - < 250 LoC
 - Includes implementation, conditional, etc.



<https://github.com/pytorch/pytorch/pull/179082>

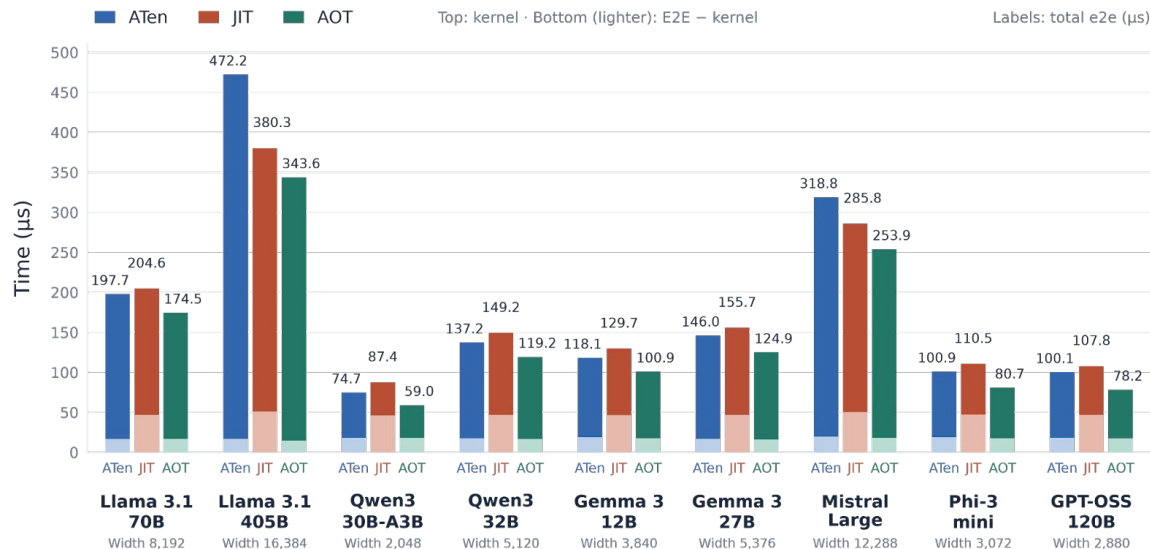
04 Details

Use Case #2 - Core Op Performance

- RMSNorm
 - Common perf moan: “{X} implementation is sooooo much faster than torch”
- Vendored a portion of the Quack lib (Tri Dao)
 - **Didn't re-implement in CUDA**
- Added first-party bindings to `aten::_fused_rmsnorm_*`
 - 1.2-2.6x faster on larger sizes
- Work done in conjunction w/NVIDIA

RMSNorm fprop: ATen vs JIT vs AOT

32,768 rows · B200 · BF16 forward · 9 selected cases from 136 cases across 17 model configurations

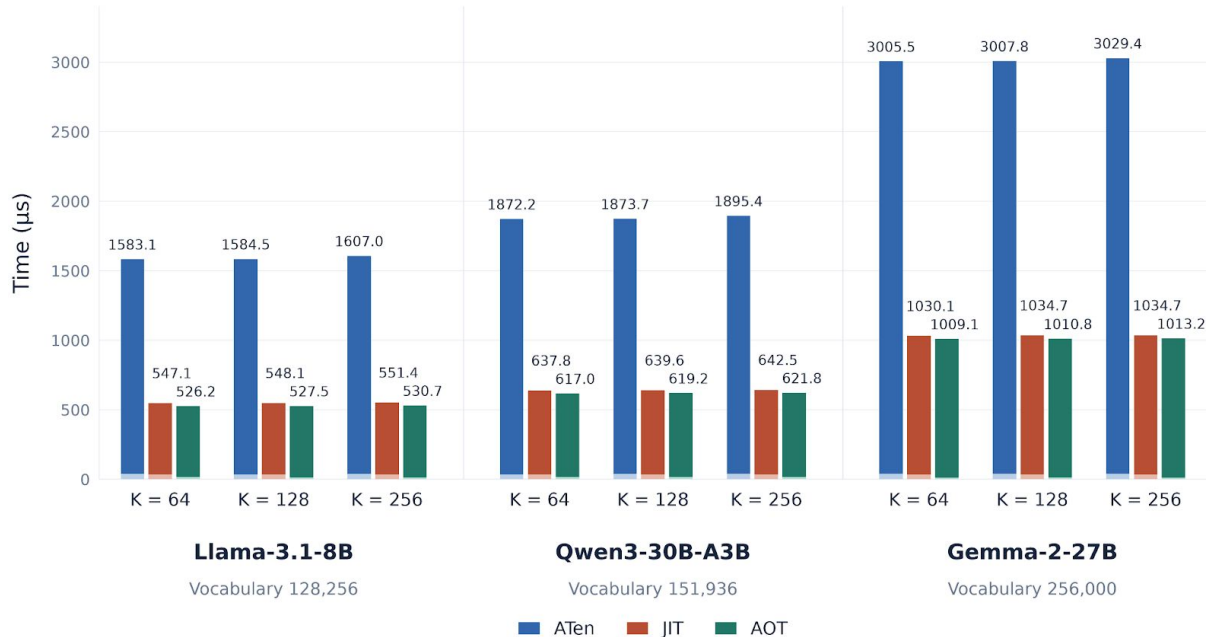


<https://github.com/pytorch/pytorch/pull/182108>

04 Details

Use Case #3 - Top-K

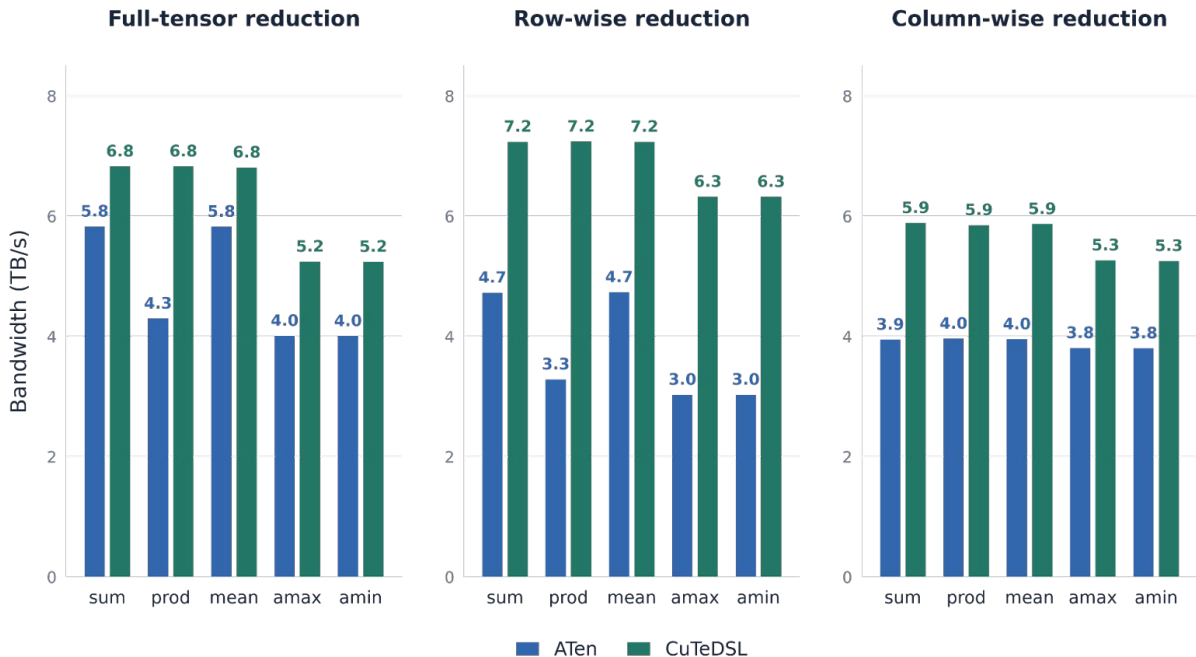
- Fast implementation from internal library
 - Written in cuteDSL
- Added as override for `aten::topk`
 - Valid for $K \in (32, 64, \dots, 1024)$
 - 1.5-4x speedup



04 Details

Use Case #4 - Clean-room rewrite

- Reduction implementation in cuteDSL (Experimental)
 - All ops/dtypes/geometries
 - ~5k LoC vs. ~15k for CUDA/C++
- B200, Median of fp16,bf16,fp32, 256MiB and 2GiB inputs



04 Details

Conclusions

- New way to contribute implementations to core
 - As first-class citizens of torch
- Opportunities throughout the perf spectrum
 - Spot-fix, major cases, complete rewrites

What's Next?

- Identifying & adding new implementations & ops
 - If you have things you want added, see me or open a PR!
- More DSLs & backends
 - FlyDSL? CPU routines?
- Just... More.
 - Of everything.